PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46

A1

(11) International Publication Number: **WO 99/44131**

(43) International Publication Date: 2 September 1999 (02.09.99)

(21) International Application Number: PCT/US99/03520

(22) International Filing Date: 18 February 1999 (18.02.99)

(30) Priority Data:

60/076,048	26 February 1998 (26.02.98)	US
09/044,917	20 March 1998 (20.03.98)	US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, MS UPAL01-521, Palo Alto, CA 94303 (US).

(72) Inventors: ARNOLD, Kenneth, C., R., C.; 7 Moon Hill Road, Lexington, MA 02173 (US). WOLLRATH, Ann, M.; 9 Northwoods Road, Groton, MA 01450 (US).

(74) Agents: GARRETT, Arthur, S.; Finnegan, Henderson, Farrabow, Garrett & Dunner, L.L.P., 1300 I Street, N.W., Washington, DC 20005-3315 (US) et al.

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

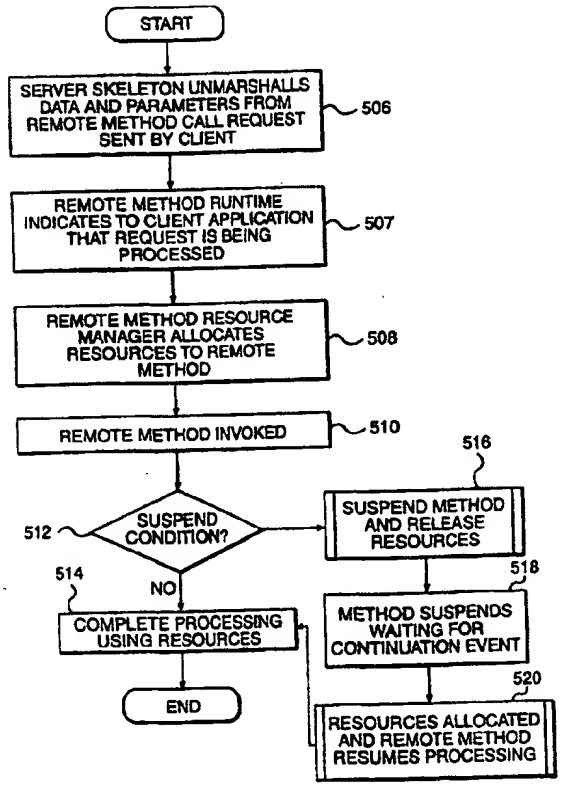
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR THE SUSPENSION AND CONTINUATION OF REMOTE PROCESSES

(57) Abstract

A method and apparatus is provided for enabling blocked remote methods to relinquish threads and other resources to other methods on a server system. In a distributed computing environment, remote methods are allocated numerous network resources but are blocked while they wait for operations, such as a write operation from another process, to complete. When enough remote methods are blocked, threads and other network resources may run out. Client systems requesting server services may experience slower response times. This method and system provides a technique for remote methods to relinquish network resources, such as threads, for other methods to use while the methods are blocked. Once the conditions causing the remote methods to block is resolved, the remote methods continue execution. This technique enables high volume client-server transaction systems to utilize threads and other resources in a distributed computing environment more efficiently.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

BNSDOCID: <WO..._3944131A1_I_>

METHOD AND APPARATUS FOR THE SUSPENSION AND CONTINUATION OF REMOTE PROCESSES

BACKGROUND OF THE INVENTION

Field of the Invention

This invention generally relates to distributed computing systems and more particularly, to a method and apparatus for the suspension and continuation of remote processes.

Related Applications

The following identified U.S. patent applications are relied upon and are incorporated by reference in this application.

Provisional U.S. Patent Application No. 60/076,048, entitled "Distributed Computing System," filed on February 26, 1998.

U.S. Patent Application No. 09/044,923, entitled "Method and System for Leasing Storage," bearing attorney docket no. 06502.0011-01000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,838, entitled "Method, Apparatus, and Product for Leasing of Delegation Certificates in a Distributed System," bearing attorney docket no. 06502.0011-02000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,834, entitled "Method, Apparatus and Product for Leasing of Group Membership in a Distributed System," bearing attorney docket no. 06502.0011-03000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,916, entitled "Leasing for Failure Detection," bearing attorney docket no. 06502.0011-04000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,933, entitled "Method for Transporting Behavior in Event Based System," bearing attorney docket no. 06502.0054-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,919, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System," bearing attorney docket no. 06502.0062-01000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,938, entitled "Methods and Apparatus for Remote Method Invocation," bearing attorney docket no. 06502.0102-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/045,652, entitled "Method and System for Deterministic Hashes to Identify Remote Methods," bearing attorney docket no. 06502.0103-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,790, entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System," bearing attorney docket no. 06502.0104-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,930, entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," bearing attorney docket no. 06502.0105-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,835, entitled "Method and System for Multi-Entry and Multi-Template Matching in a Database," bearing attorney docket no. 06502.0107-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,839, entitled "Method and System for In-Place Modifications in a Database," bearing attorney docket no. 06502.0108, and filed on the same date herewith.

U.S. Patent Application No. 09/044,945, entitled "Method and System for Typesafe Attribute Matching in a Database," bearing attorney docket no. 06502.0109-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,931, entitled "Dynamic Lookup Service in a Distributed System," bearing attorney docket no. 06502.0110-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,939, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," bearing attorney docket no. 06502.0112-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,826, entitled "Method and System for Facilitating Access to a Lookup Service," bearing attorney docket no. 06502.0113-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,932, entitled "Apparatus and Method for Dynamically Verifying Information in a Distributed System," bearing attorney docket no. 06502.0114-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network," bearing attorney docket no. 06502.0115-00000, and filed on February 26, 1998.

U.S. Patent Application No. 09/044,936, entitled "An Interactive Design Tool for Persistent Shared Memory Spaces," bearing attorney docket no. 06502.0116-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,934, entitled "Polymorphic Token-Based Control," bearing attorney docket no. 06502.0117-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,915, entitled "Stack-Based Access Control," bearing attorney docket no. 06502.0118-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,944, entitled "Stack-Based Security Requirements," bearing attorney docket no. 06502.0119-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,837, entitled "Per-Method Designation of Security Requirements," bearing attorney docket no. 06502.0120-00000, and filed on the same date herewith.

Description of the Related Art

A typical distributed computing system makes the services and computing power of many servers available to many different clients over a network. Typically, a client machine accesses processing capabilities on a server machine using a remote procedure call (RPC) system. The RPC system processes the request on the remote machine and returns the desired results to the requesting client. The network used to transmit the request and return the results can be a local area network (LAN), a wide area network (WAN), and can also include the Internet. Sophisticated distributed computing applications on the Internet offer electronic commerce (e-commerce), telecommuting, and interactive entertainment services around the world using this client-server arrangement.

As the distributed computing paradigm grows in usage and popularity it is increasingly important that resources on the server systems are available to fulfill requests made by the client systems. Each request from a client generally causes a server process to designate resources,

including one or more threads, to process the requests. A thread, sometimes called a lightweight process, is a separate sequence of instructions within a process having a separate flow of control. The thread must carve out resources from the system as needed to fulfill the particular request. If resources such as memory and data are available, multiple threads can be executed in parallel to fulfill multiple tasks.

Thread schedulers can be used by the process spawning the threads to coordinate parallel execution of the threads based on a thread's priority, state of execution (i.e. sleep, alive, dead, running), and dependencies among the various threads. A thread scheduler on a single processor system distributes the processor's computing power among many threads to provide the illusion that the threads are actually running in parallel. There are many different scheduling techniques which can be used including first-come-first-served, shortest-thread-first, priority scheduling, and preemptive scheduling techniques such as round-robin. Hybrid scheduling techniques which combine these techniques can also be used as needed by the particular implementation. On multiprocessor systems, schedulers associate different threads with different processors to execute threads in parallel and take advantage of the added computing power.

Unfortunately, if these resources are not immediately available, the thread can not continue execution and is blocked from further processing. These blocked threads of execution hold on to server resources, such as memory, as well as the data and control structures associated with the threads themselves. Eventually, the server may run out of threads to allocate incoming client requests. Incoming client requests may be refused and the server will be effectively removed from the distributed computing environment. This blocking scenario can also reduce a servers ability to service existing requests due to overhead associated with denying service to the incoming calls.

Present distributed computing systems are not designed to address this problem of allocating threads. These systems do not release threads and associated resources when a remote server process is blocked waiting for a resource or particular event. Consequently, transaction intensive distributed computing environments can suffer from the blocking scenario described above. For example, assume a server process receives multiple requests to download a file from multiple clients over the Internet. The server process receives multiple threads from the server operating system to process the requests in parallel but the file requested is locked by another process and is not available. A conventional system would block further processing on each

PNSDOCID: <WO__9944131A1_I_>

thread and wait for the file to be unlocked. The threads on the server would remain idle even though other processes could utilize the thread resources to process other tasks. When the number of threads on a server system are depleted, the server process will deny service to additional clients. Eventually, the server system will have difficulty processing general tasks.

On many distributed computing systems, the inability to allocate threads and other resources can negatively impact overall processing throughput. Even the high-speed bandwidth available on a distributed computing network will go unused if threads and other resources are not allocated efficiently on the server system.

Based on the above limitations found in conventional systems, it is desirable to improve the allocation of threads and other resources used in a distributed computing environment.

SUMMARY OF THE INVENTION

Consistent with the present invention, as embodied and broadly described herein, a method and apparatus for enabling a remote method to suspend processing and relinquish resources to the server system comprises receiving a request from a remote method call on a client system. The remote method is allocated system resources from the server system and invoked. One system level type of resource is a thread. The method determines if any general resources required for processing the remote method are presently not available. A general resource can be memory, disk storage space, data, or any resource that a system resource may depend on. The remote method is suspended from further processing and system resources are relinquished to the server system when the remote method depends on a general resource which is not available.

Another method consistent with the present invention, enables a previously suspended remote method to continue processing on the server system and generate a result for a client application. This method comprises receiving an indication that a continuation event associated with a suspended remote method has occurred. System resources and general resources are allocated to the remote method in preparation to continue processing the remote method. The remote method utilizes the combined allocated resources to continue execution and generate results. These results are transmitted from the server system to the client application on the client system using a remote procedure call (RPC) system such as remote method invocation (RMI).

RNSDOCID: <WO___9944131A1_I_>

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an embodiment of the invention and, together with the description, serve to explain the advantages, and principles of the invention.

In the drawings:

FIG. 1. illustrates a network suitable for use with methods and systems consistent with the present invention;

FIG. 2 is block diagram of a computer system suitable for use with methods and systems consistent with the present invention;

FIG. 3 is a block diagram of a client-server networking environment suitable for use with methods and systems consistent with the present invention;

FIG. 4 is a block diagram of the subsystems used to suspend and continue processing of remote method calls consistent with methods and systems of the present invention;

FIG. 5 is a flow chart of the steps performed to suspend and continue a remote method call consistent with methods and systems of the present invention;

FIG. 6 is a flow chart of the steps performed to suspend a remote method call consistent with methods and systems of the present invention; and

FIG. 7 is a flow chart of the steps performed to continue a remote method call consistent with methods and systems of the present invention.

DETAILED DESCRIPTION

INTRODUCTION

Reference will now be made in detail to an implementation of the present invention as illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings and the following description to refer to the same or like parts.

A system designed consistent with the present invention assumes that a computer system can host either client or server functions. The roles each computer assumes in a client-server system depends on the specific call being made between the client and the server. For example, a client process typically requests a service generated by a server process located on a remote 5machine. Conversely, a server process is located on a machine which receives and services a

clients requests. Therefore, the same computer system can act as a client when requesting a service and a server when fulfilling a request for a service.

Systems consistent with the present invention address shortcomings of the prior art and provide a method and apparatus for the suspension and continuation of remote processes. In the past, remote procedure call (RPC) systems did not enable a remote process to release resources while the remote process was blocked waiting for an event to occur or a resource to be released. This technique kept the connection between the client process and server process active but left important resources such as threads, memory, and secondary storage idle. In contrast, systems designed consistent with the present invention allow a remote server process blocked from further processing to relinquish threads and other resources while not dropping the connection between the client and server systems. This novel technique permits other processes to utilize server resources even when a remote process is blocked. Specifically, this prevents a remote server process from denying clients access to the system. Consequently, embodiments of the present invention use threads and other resources more efficiently which increases the effective throughput of a distributed computing system.

In addition, systems consistent with the present invention are also advantageous in that they are compatible with clients on existing client-server systems. This is especially important in heterogeneous networked computing environments such as the Internet. Because the server is modified to allocate and deallocate resources more efficiently, the present invention does not influence the design and operation of the client system. Accordingly, a client would not require modification to work with a system of the present design.

OVERVIEW OF THE DISTRIBUTED SYSTEM

Methods and systems consistent with the present invention operate in a distributed system ("the exemplary distributed system") with various components, including both hardware and software. The exemplary distributed system (1) allows users of the system to share services and resources over a network of many devices; (2) provides programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplifies the task of administering the distributed system. To accomplish these goals, the exemplary distributed system utilizes the Java™ programming environment to allow both code and data to be moved from device to device in a seamless manner. Accordingly, the exemplary distributed system is layered on top of the Java programming environment and exploits the

BNSDOCID: <WO___9944131A1_I_>

characteristics of this environment, including the security offered by it and the strong typing provided by it. The Java programming environment is more clearly described in Jaworski, Java 1.1 Developer's Guide, Sams.net (1997), which is incorporated herein by reference.

In the exemplary distributed system, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single system, the exemplary distributed system provides the simplicity of access and the power of sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. The exemplary distributed system may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Within the exemplary distributed system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as applications or utilities; information, such as databases and files; and users of the system.

Both users and devices may join a Djinn. When joining a Djinn, the user or device adds zero or more services to the Djinn and may access, subject to security constraints, any one of the services it contains. Thus, devices and users federate into a Djinn to share access to its services. The services of the Djinn appear programmatically as objects of the Java programming environment, which may include other objects, software components written in different programming languages, or hardware devices. A service has an interface defining the operations that can be requested of that service, and the type of the service determines the interfaces that make up that service.

Fig. 1 depicts the exemplary distributed system 100 containing a computer 102, a computer 104, and a device 106 interconnected by a network 108. The device 106 may be any of a number of devices, such as a printer, fax machine, storage device, computer, or other devices. The network 108 may be a local area network, wide area network, or the Internet. Although only two computers and one device are depicted as comprising the exemplary

distributed system 100, one skilled in the art will appreciate that the exemplary distributed system 100 may include additional computers or devices.

Fig. 2 depicts the computer 102 in greater detail to show a number of the software components of the exemplary distributed system 100. One skilled in the art will appreciate that computer 104 or device 106 may be similarly configured. Computer 102 includes a memory 202, a secondary storage device 204, a central processing unit (CPU) 206, an input device 208, and a video display 210. The memory 202 includes a lookup service 212, a discovery server 214, and a Java™ runtime system 216. The Java runtime system 216 includes the Java™ remote method invocation system (RMI) 218 and a Java™ virtual machine 220. The secondary storage device 204 includes a Java™ space 222.

As mentioned above, the exemplary distributed system 100 is based on the Java programming environment and thus makes use of the Java runtime system 216. The Java runtime system 216 includes the Java™ API, allowing programs running on top of the Java runtime system to access, in a platform-independent manner, various system functions, including windowing capabilities and networking capabilities of the host operating system. Since the Java API provides a single common API across all operating systems to which the Java runtime system 216 is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. The Java runtime system 216 is provided as part of the Java™ software development kit available from Sun Microsystems of Mountain View, CA.

The Java virtual machine 220 also facilitates platform independence. The Java virtual machine 220 acts like an abstract computing machine, receiving instructions from programs in the form of byte codes and interpreting these byte codes by dynamically converting them into a form for execution, such as object code, and executing them. RMI 218 facilitates remote method invocation by allowing objects executing on one computer or device to invoke methods of an object on another computer or device. Both RMI and the Java virtual machine are also provided as part of the Java software development kit.

The lookup service 212 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within the exemplary distributed system 100. The lookup service 212 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the

ENSDOCID: <WO___9944131A1_I_>

corresponding service. The lookup service 212 and its access are described in greater detail in co-pending U.S. Patent Application No. 09/044,826, entitled "Method and System for Facilitating Access to a Lookup Service," which has previously been incorporated by reference.

The discovery server 214 detects when a new device is added to the exemplary distributed system 100, during a process known as boot and join or discovery, and when such a new device is detected, the discovery server passes a reference to the lookup service 212 to the new device, so that the new device may register its services with the lookup service and become a member of the Djinn. After registration, the new device becomes a member of the Djinn, and as a result, it may access all the services contained in the lookup service 212. The process of boot and join is described in greater detail in co-pending U.S. Patent Application No. 09/044,939, entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which has previously been incorporated by reference.

The Java space 222 is an object repository used by programs within the exemplary distributed system 100 to store objects. Programs use the Java space 222 to store objects persistently as well as to make them accessible to other devices within the exemplary distributed system. Java spaces are described in greater detail in co-pending U.S. Patent Application No. 08/971,529, entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, filed on November 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that the exemplary distributed system 100 may contain many lookup services, discovery servers, and Java spaces.

Although systems and methods consistent with the present invention are described as operating in the exemplary distributed system and the Java programming environment, one skilled in the art will appreciate that the present invention can be practiced in other systems and other programming environments. Additionally, although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from the Internet; or other forms of RAM or ROM. Sun, Sun Microsystems, the SunLogo, Java, and Java-based trademarks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries.

BNSDOCID: <WO__9944131A1_I_>

EXEMPLARY CLIENT-SERVER SYSTEM

FIG. 3 depicts an exemplary client-server system consistent with the present invention and exemplary distributed system 100. Accordingly, client-server system 300 consists of a client computer 302, also referred to as client 302, a server computer 312, also referred to as server 312, and a network 310 coupled between client 302 and server 312. This particular client-server system can be implemented using the Java™ object oriented language and as an enhancement to RMI 218. However, those skilled in the art will appreciate that similar systems consistent with the present invention can be implemented using a general remote procedure call (RPC) system and other object and non-object oriented languages.

Client 302 includes a client application 304 having a remote method call 306, a remote stub 308, and a remote method runtime 309. Client application 304 is typically software developed by a user and includes remote method call 306 for invoking a process on server 312. For example, client application 304 can be a Java™ application written in the Java™ programming language. Remote method call 306 is implemented using an RPC mechanism such as RMI.

Remote method stub 308 marshals data and parameters provided by remote method call 306. The data and parameters are arranged in a predetermined format that can be unmarshalled by a remote method skeleton 315 on server 312. Remote method runtime 309 tracks the status of processes associated with remote method call 306 as they are processed on server 312. Remote method runtime 309 also determines whether the communication link between client application 304 and server 312 is up or has been disconnected. Remote method runtime 309 can query server 312 for the status of the link. If no response is made in a reasonable period of time or server 312 indicates the link is down, remote method runtime 309 notifies client application 304 that the remote method call has terminated.

Network 310 provides a communication link between client 302 and server 312. Network 310 can be the Internet or a corporate or campus-wide intranet. Network 310 can use TCP/IP or any other network protocols including Novell Netware, AppleTalk, X.25, or any other network capable of supporting an RPC system such as RMI.

Server 312 includes a corresponding remote method runtime 314 and remote method skeleton 315. In contrast to client 302, server 312 also includes a general resource manager 316, an event handler 317, a remote method resource manager 322, a remote event handler 323, and

BNSDOCID: <WO__9944131A1_I_>

numerous remote method resources 324. Server 312 also includes a remote object A 318 and a remote object B 320. Each remote object is associated with a number of methods (not shown) which client application 304 can invoke using remote method call 306. Alternative configurations of server 312 may include any number of remote objects for performing remote methods as required by the particular system.

Remote method runtime 314 is responsible for keeping client 302 informed of the remote method execution status. Remote method runtime 314 provides information to client 302 indicating that the remote method is processing data. Consistent with the present invention, the processing status is not interrupted even when a remote method is suspended. Instead, remote method runtime 314 maintains the connection with client 302 until the remote method completes processing the requested task. Remote method runtime 314 also indicates to client 302 when a remote method has terminated abnormally or in error.

The method determines if any general resources required for processing the remote method are presently not available. A general resource can be memory, disk storage space, data, or any resource that a system resource may depend on.

Remote method skeleton 315 is responsible for unmarshalling data and parameters transmitted over network 310. The parameters and data are used as arguments for executing a remote method on server 312.

General resource manager 316 and event handler 317 manage resources used by local processes and methods executed on server 312. Local methods executed on server 312 look to general resource manager 316 and event handler 317 to coordinate allocation and deallocation of general resources. These general resources can include primary storage, such as memory, or secondary storage, such as disk and tape drives. Unlike system resources discussed below, general resources are typically not used for fulfilling remote method requests. Event handler 317 detects events associated with local processes and, therefore, the details of event handler 317 are not included in this specification. Essentially, general resource manager 316 and event handler 317 are dedicated to managing resources associated with those processes and methods which are not being invoked remotely from a client such as client 302.

In contrast, remote method resource manager 322 and remote event handler 323 are responsible for allocating and deallocating remote method resources 324 as needed by remote methods. Remote method resources 324 can be considered a system resource since they enable

a method to utilize a system level resource such as networking. Often, the system resources will have dependencies on the general resources mentioned above.

Remote method resource manager 322 transfers remote method resources 324 between remote methods associated with remote object 318, remote object 320, and other objects (not shown). Remote event handler 323 detects when resources are released that a remote method needs to process a particular task. Transferring these remote method resources is facilitated utilizing an implementation of the present invention as discussed below.

SUSPEND AND CONTINUATION OF REMOTE METHODS

A suspend method is invoked when a remote method is about to block. This typically occurs just before a remote method is blocked waiting for a resource to become available. The suspend method marks the remote method as suspended and a remote method resource manager returns threads and other resources back to the server system. When the resources become available, a remote method resource manager and a continue method work together to allocate the threads and other remote method resources to the suspended remote method. Eventually, a remote method runtime invokes the suspended remote method so that it may continue processing. For example, a remote method waiting for a write operation enters a ready to block state and relinquishes threads and other resources by invoking a suspend operation. The suspend operation marks the remote method as suspended and the remote method resource manager returns the threads to a thread pool associated with the server system. Once the write operation occurs, a continue method marks the suspended remote method as runnable and the remote method resource manager allocates threads and to other resources back to the suspended remote process. A remote method runtime invokes the previously suspended remote method which enables the remote method to read the data.

Figure 4 is a block diagram illustrating the essential software subsystems used to suspend and continue processing a remote method. These software subsystems include remote method resources 324, remote method resource manager 322, and exemplary remote object A 318 having remote method 416 and execution state 418.

Remote method resources 324 in Fig. 4 includes a thread pool 402 having threads in-use 404, available threads 406, and an RPC state 408. Threads in-use 404 contains references to threads currently being used by remote methods while available threads 406 contains threads

currently available for use by remote methods. RPC state 408 is kept in remote method resources 324 to store information used by an RPC system, such as RMI, when a remote method is suspended. This information can include information the RPC system to continue processing a suspended remote method and return results to the client. In an alternative embodiment, remote method resources 324 could also include other resources other than threads. These other network resources could include primary storage, secondary storage and any other resource used in conjunction with processing remote methods.

Remote method resource manager 322 includes a suspend method 410, a continuation method 412, and a state store 414. Suspend method 410 obtains an execution state 418 from remote method 416 and a RPC state 408 from remote method resources 324. This state information is stored in state store 414 before remote method 416 is suspended. Typically, suspend method 410 marks a remote method 416 as suspended when remote method 416 indicates that it is about to be blocked and that it has threads and other resources. Eventually, remote method resource manager 322 returns the threads and other resources to server 312 and remote method 416 is suspended from further processing. For example, assume remote method 416 is attempting to read data from a queue which is temporarily empty. When remote method 416 detects the queue is empty, remote method 416 will invoke suspend method 410 to initiate the suspension process.

Continue method 412 is the companion to suspend method 410. Continue method 412 is typically invoked when a resource is available or a particular event has occurred. For example, writing data to a particular queue can trigger a continuation event which can invoke continue method 412. Continue method 412 locates the suspended remote method waiting on the resource and marks it as runnable. Eventually, remote method resource manager 322 allocates threads and other resources to the previously suspended remote method. Execution state 418 and RPC state 408 stored in state store 414 are used to ensure that remote method 416 continues processing at the appropriate point prior to being suspended.

In operation, the suspend and continuation methods are used together to manage threads and other resources on server system 312. FIG. 5 is a flow chart indicating the steps performed to suspend and continue a remote call in consistent with methods and systems of the present invention.

BNSDOCID: <WO__9944131A1_I_>

Initially, the server 312 receives a request from remote method call 306 on client 302 to process remote method 416. Accordingly, remote method skeleton 315 unmarshalls the data and parameters transmitted in the request (step 506). After the data and parameters are decoded by remote method skeleton 315 they are passed to remote method 416.

Remote method runtime 314 on server 312 indicates to remote method runtime 309 on client 302 that server 312 has received the request to invoke remote method 416 and is processing the request (step 507). Client application 304 continues to receive an indication that server 312 is processing the request even if remote method 416 is suspended and threads and other resources are relinquished.

Remote method resource manager 322 allocates the threads and other resources to remote method 416 which is about to be invoked (step 508). Threads allocated to remote method 416 are taken from available threads 406 in thread pool 402. Multiple threads can be used to process several remote methods or tasks in parallel. Assuming the threads and other resources are available, remote method runtime 314 invokes remote method 416 on behalf of client application 304 (step 510). If the remote method resource manager cannot allocate a thread or other resources to remote method 416, it is put on an execution queue pending release of a thread or other resources from another process.

Remote method 416 includes instructions for determining if a suspend condition exists (step 512). These instructions also include information to determine when a continuation condition exists. A suspend condition occurs when remote method 416 depends on a resource which is not available or an event which has not yet occurred. In contrast, the continuation event occurs when the resource is available or the event occurs. For a suspend condition example, assume remote method 416 is sampling data points and pauses a long time interval between each sampling. During this long pause, remote method 416 may block waiting for a timer event to indicate the end of the next time interval.

When the suspend condition is detected, remote method 416 relinquishes resources and registers continuation instructions with a remote event handler 323 to monitor certain resources and events (step 516). Next, remote method 416 is suspended from further processing and awaits the particular continuation event in order to continue processing (step 518).

When the continuation event occurs, resources are allocated to the remote method and the remote method resumes processing (step 520). A continuation event is an event generated

BNSDOCID <WO___9944131A1_I_>

when a resource becomes available or an event occurs that a suspended remote method depends on for further processing. Remote event handler 323 processes the continuation instructions registered by suspended remote method 416. The remote method completes the task and returns the results back to client application 304 (step 514). It should be understood that a remote method can be suspended and continued many times before completing a task and returning results to the client.

SUSPENSION OF REMOTE METHODS

FIG. 6 is a flow chart of the steps performed to suspend a remote call consistent with methods and systems of the present invention. Initially, remote method 416 detects that a suspend event has occurred and elects to relinquish threads and other resources (step 602). Remote method is marked as being suspended by suspend method 410. Before being suspended, remote method 416 provides remote method resource manager 322 with execution state 418, RPC state 408, and relinquishes the threads and other resources (step 604). Remote method resource manager 322 places these threads back into available threads 406. Alternatively, remote method 416 can choose to suspend processing without relinquishing any threads or resources.

Execution state 418 records state information related to remote method 418 at the time it is suspended including local variables, program counter, and any other information related to remote method 416. As mentioned above, RPC state 408 records state information associated with the RPC system, such as RMI, when remote method 416 is suspended. RPC state 408 enables the RPC system to communicate with a client and return results when remote method 416 continues execution.

Next, remote method resource manager 322 stores RPC state 408 and execution state 418 (step 608). Remote method resource manager 322 stores this state information to continue processing a suspended remote method at some time in the future. Remote method 416 also registers continuation instructions with remote event handler 323 which monitors certain resources and events. Generally, the continuation instructions are used for preprocessing data and contacting the appropriate suspended process. Last, remote method 416 is blocked from further processing and waits for a continuation event to occur (step 610).

CONTINUATION OF REMOTE METHODS

FIG. 7 is a flow chart of the steps performed to continue a previously suspended remote method, such as remote method 41 with methods and systems consistent with the present invention. Typically, each step of this process occurs asynchronously as the particular conditions are met.

Initially, remote event handler 323 receives an indication that a particular continuation event has occurred. Remote event handler 323 associates the continuation event with a particular remote method (step 702) and invokes the corresponding continuation instructions. The continuation event is an event generated when a resource becomes available or an event occurs which the suspended remote method depends on for further processing. For example, a continuation event can occur when information is written to a queue that a suspended remote method was waiting to read.

The continuation instructions invoke continue method 412 which marks the suspended remote method 416 as a runnable process. Eventually, remote method resource manager 322 discovers the status of suspended method 416 which indicates the remote method is now runnable and can continue processing (step 704). Step 704 is typically an asynchronous process that occurs when remote method resource manager checks for status on the suspended process and not necessarily when status of the remote process is changed to runnable. Remote method resource manager 322 allocates the resources, such as threads, to remote method 416 (step 706).

Remote method runtime 314 loads execution state 418 and RPC state 408 from state store 414. Execution state 418 prepares remote method 416 to continue processing at the point it left off prior to being suspended (step 708). To implement this in software, remote method 416 may be several pieces of interrelated code which each start where the previous code section stopped prior to being suspended. Alternatively, remote method 416 can be implemented as a single code segment with execution starting at different points in the code segment. As previously mentioned, RPC state 408 enables the RPC system, such as RMI 218, to continue processing and return results to the proper remote method call 306.

Remote method runtime 314 enables remote method 416 to continue execution. Remote method 416 then continues processing on the server using the assigned threads and other resources (step 710). Eventually, results are generated by remote method 416 and are provided to remote method skeleton 315 for encoding and packaging (step 712). These encoded results

BNSDOCID: <WO__9944131A1_I_>

are passed over network 310 to remote method stub 308 where they are decoded and provided to remote method call 306 (step 714).

While specific embodiments have been described herein for purposes of illustration, various modifications may be made without departing from the spirit and scope of the invention. Accordingly, the invention is not limited to the above described embodiments, but instead is defined by the appended claims in light of their full scope of equivalents.

WHAT IS CLAIMED IS:

1. A method performed on a server system having resources which is operatively coupled to a client system over a network and enables a remote method to suspend processing and relinquish corresponding resources to the server system, the method comprising the steps of:

receiving a request from a client system with a remote method call transmitted using a remote procedure call (RPC) system which further includes the substeps of:

indicating to the client system that the server system has received the request to invoke a remote method;

allocating resources to the remote method;

invoking the remote method;

determining if the remote method depends on any resources which are not available or an event which has not occurred yet;

suspending the remote method from further processing and relinquishing resources from the remote method when the remote method depends on a resource that is not available or an event that has not occurred yet, which further includes the substeps of,

relinquishing resources previously allocated to the remote method system for use by other processes and methods on the server system;

providing an execution state associated with the remote method and a remote procedure call (RPC) state associated with the remote method call to continue processing the remote method at a subsequent time period; and

blocking the remote method from further processing until the resource which was not available is released and made available to the remote method or the event that caused the suspension of the remote method.

A method performed on a server system having a plurality of resources which is operatively coupled to a client system over a network, for enabling a previously suspended remote method to continue processing on the server system, the method comprising the steps of:

receiving an indication that a continuation event associated with a suspended remote method has occurred wherein the continuation event is generated when a required resource for the remote method becomes available or an event occurs that previously caused the suspension of the remote method;

allocating resources to the remote method in preparation to continue processing, which further comprises the substeps of:

indicating to a remote method resource manager that the remote method can continue processing, and

distributing resources controlled by a remote method resource manager to the remote method;

continuing execution of the remote method utilizing the allocated resources, including the substep of:

initializing the remote method with an execution state and a remote procedure call (RPC) system with a RPC state wherein the execution state includes information associated with the remote method before it was suspended and the RPC state includes information associated with the remote method call;

generating results from the remote method; and

transmitting the results from the server system to the client application on the client system.

A method performed on a server system for enabling a remote method to suspend processing and relinquish resources to the server system which is operatively coupled to a client system over a network, the method comprising the steps of:

receiving a request from a remote method call on a client system;

allocating system resources to the remote method associated with the server system;

invoking the remote method associated with the server system;

determining an availability of general resources required for processing the remote method; and

suspending the remote method from further processing when the remote method depends on a general resource which is not available.

The method of Claim 3 wherein the server system includes a processor, a primary storage device, a secondary storage device, a display device, and an input/output mechanism.

The method of Claim 3 wherein the system resources include threads.

The method of Claim 3 wherein the general resources include data.

The method of Claim 3 wherein the request is transmitted using a remote method invocation (RMI) system.

The method of Claim 3 wherein the suspending step further includes the substeps of,

relinquishing system resources and general resources to the server system; and

blocking the remote method from further processing until the resource which was not available is released and made available to the remote method.

The method of Claim 8 wherein the suspending step further includes the substeps of providing an execution state associated with the remote method and a remote procedure call (RPC) state associated with the remote method call to continue processing the remote method at a subsequent time period.

A method performed on a server system for enabling a previously suspended remote method to continue processing on the server system and generate a result for a client application on a client system which is operatively coupled to the server system over a network, the method comprising the steps of:

receiving indication that a continuation event associated with a suspended remote method has occurred;

allocating system resources and general resources to the remote method in preparation to continue processing:

continuing execution of the remote method utilizing the allocated resources;

generating results from the remote method; and

transmitting the results from the server system to the client application on the client system.

The method of Claim 10 wherein the server system includes a processor, a primary storage device, a secondary storage device, a display device, and an input/output mechanism.

The method of Claim 10 wherein the system resources include threads.

The method of Claim 10 wherein the general resources includes data.

The method of Claim 10 wherein the step of continuing execution of the remote method further includes the substeps of:

initializing the remote method with an execution state and a remote procedure call (RPC) system with a RPC state wherein the execution state includes information associated with the remote method before it was suspended and the RPC state includes information associated with the remote method call; and

generating results from the remote method.

The method of Claim 10 wherein the transmitting step uses a remote method invocation (RMI) system.

A computer-readable medium containing instructions which enables a remote method executed on a computer server system to suspend processing and relinquish resources to the computer server system by:

receiving a request from a remote method call on a client system;

allocating system resources to the remote method associated with the server system;

invoking the remote method associated with the server system;

determining an availability of general resources required for processing the remote method; and

suspending the remote method from further processing when the remote method depends on a general resource which is not available.

The computer-readable medium of Claim 16 wherein the system resources include threads.

The computer-readable product of Claim 16 wherein the general resources include data.

The computer-readable medium of Claim 16 wherein the request is transmitted using an remote method invocation (RMI) system.

The computer-readable medium of Claim 16 wherein the suspending is further performed by,

relinquishing module configured to relinquish system resources and general resources to the server system; and

a blocking module configured to block the remote method from further processing until the resource which was not available is released and made available to the remote method.

A computer-readable medium containing instructions which enables a previously suspended remote method to continue processing on a server system and generate a result for a client application on a client system by:

receiving indication that a continuation event associated with a suspended remote method has occurred;

allocating system resources and general resources to the remote method in preparation to continue processing:

continuing execution of the remote method utilizing the allocated resources;

generating results from the remote method; and

transmitting the results from the server system to the client application on the client system.

BNSDOCID: <WO___9944131A1_I_>

The computer-readable medium of Claim 21 wherein the system resources include threads.

The computer-readable product of Claim 21 wherein the general resources include data.

The computer-readable product of Claim 21 wherein the transmitting uses a remote method invocation (RMI) system.

An apparatus for enabling a remote method to suspend processing and relinquish resources to a server system which is operatively coupled to a client system over a network, comprising:

a receiver module configured to receive a request from a remote method call on a client system;

an allocating module configured to allocate system resources to the remote method associated with the server system;

an invoking module configured to invoke the remote method associated with the server system;

a determining module configured to determine if any general resources required for processing the remote method are not available; and

a suspending module configured to suspend the remote method from further processing when the remote method depends on a general resource which is not available.

The apparatus of Claim 25 wherein the server system includes a processor, a primary storage device, a secondary storage device, a display device, and an input/output mechanism.

The apparatus of Claim 25 wherein the system resources include threads.

The apparatus of Claim 25 wherein the general resources include data.

The apparatus of Claim 25 wherein the request is transmitted using a remote method invocation (RMI) system.

The apparatus of Claim 25 wherein the suspending module step further comprises,

a relinquishing module configured to relinquish system resources and general resources to the server system; and

a blocking module configured to block the remote method from further processing until the resource which was not available is released and made available to the remote method.

An apparatus for enabling a previously suspended remote method to continue processing on a server system and generate a result for a client application on a client system which is operatively coupled to the server system over a network, the apparatus comprising:

a receiver module to receive an indication that a continuation event associated with a suspended remote method has occurred;

an allocating module configured to allocate system resources and general resources to the remote method in preparation to continue processing:

a continuing module configured to continue execution of the remote method utilizing the allocated resources;

a generating module configured to generate results from the remote method; and

a transmitting module configured to transmit the results from the server system to the client application on the client system.

2. The apparatus of Claim 31 wherein the server system includes a processor, a primary storage device, a secondary storage device, a display device, and an input/output mechanism.

The apparatus of Claim 31 wherein the system resources include threads.

The apparatus of Claim 31 wherein the general resources includes data.

BNSDOCID: <WO___9944131A1_I_>

3. The apparatus of Claim 31 wherein the transmitting module uses a remote method invocation (RMI) system.

An apparatus coupled to a server system for enabling a remote method to suspend processing and relinquish resources to the server system which is operatively coupled to a client system over a network, the apparatus comprising:

a means for receiving a request from a remote method call on a client system;

a means for allocating system resources to the remote method associated with the server system;

a means for invoking the remote method associated with the server system;

a means for determining an availability of general resources required for processing the remote method; and

a means for suspending the remote method from further processing when the remote method depends on a general resource which is not available.

The apparatus of Claim 36 further comprising a client system operatively coupled to the server system.

An apparatus coupled to a server system for enabling a previously suspended remote method to continue processing on the server system and generate a result for a client application on a client system which is operatively coupled to the server system over a network, the method comprising the steps of:

a means for receiving indication that a continuation event associated with a suspended remote method has occurred;

a means for allocating system resources and general resources to the remote method in preparation to continue processing:

a means for continuing execution of the remote method utilizing the allocated resources;

a means for generating results from the remote method; and

a means for transmitting the results from the server system to the client application on the client system.

The apparatus of Claim 38 further comprising a client system operatively coupled to the server system.

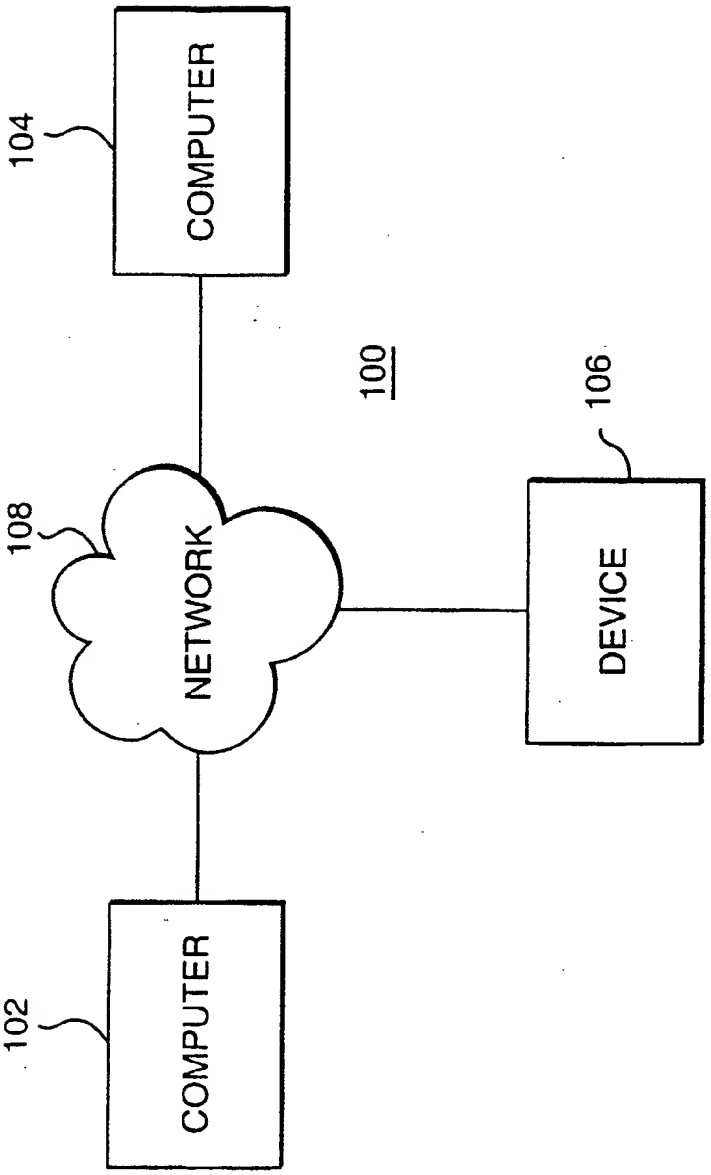


FIG. 1

BNSDOCID: <WO___9944131A1_I_>

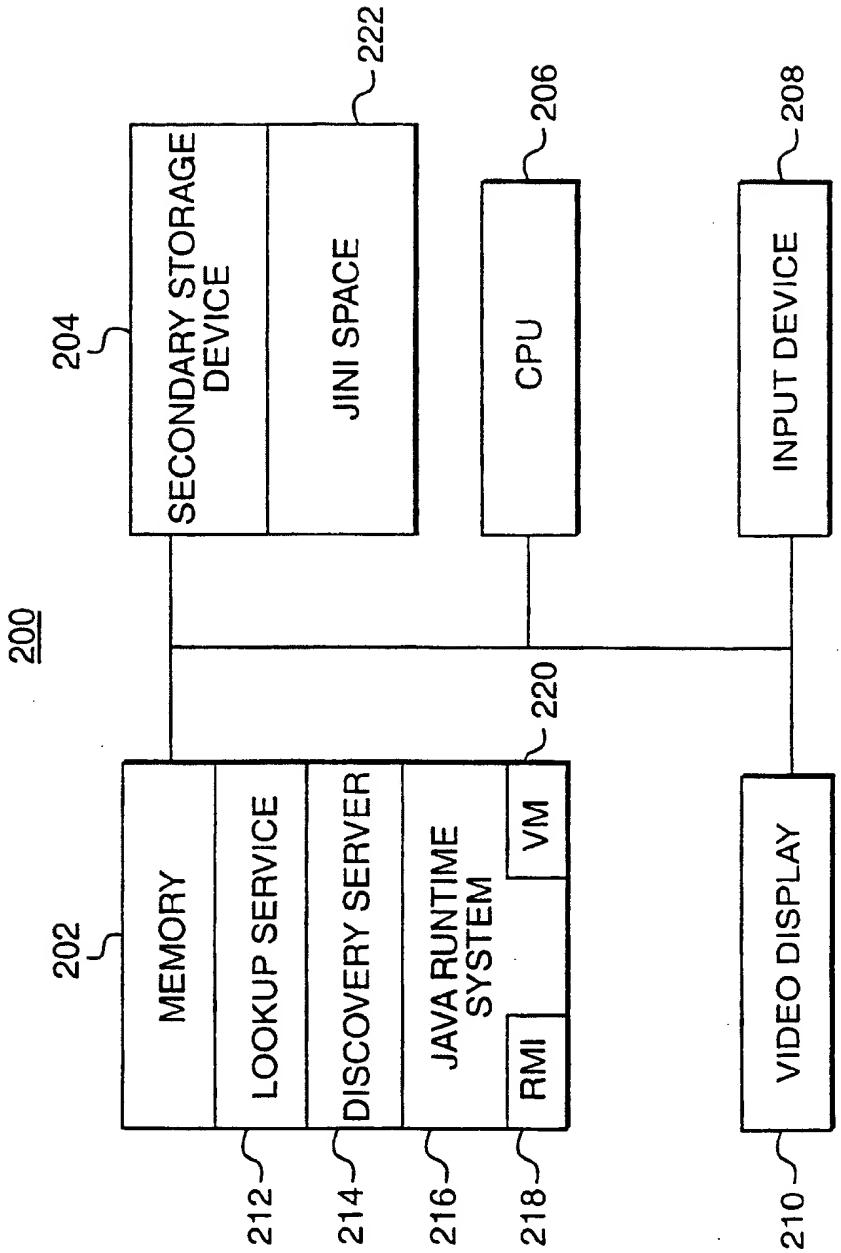


FIG. 2

BNSDOCID: <WO__9944131A1_I_>

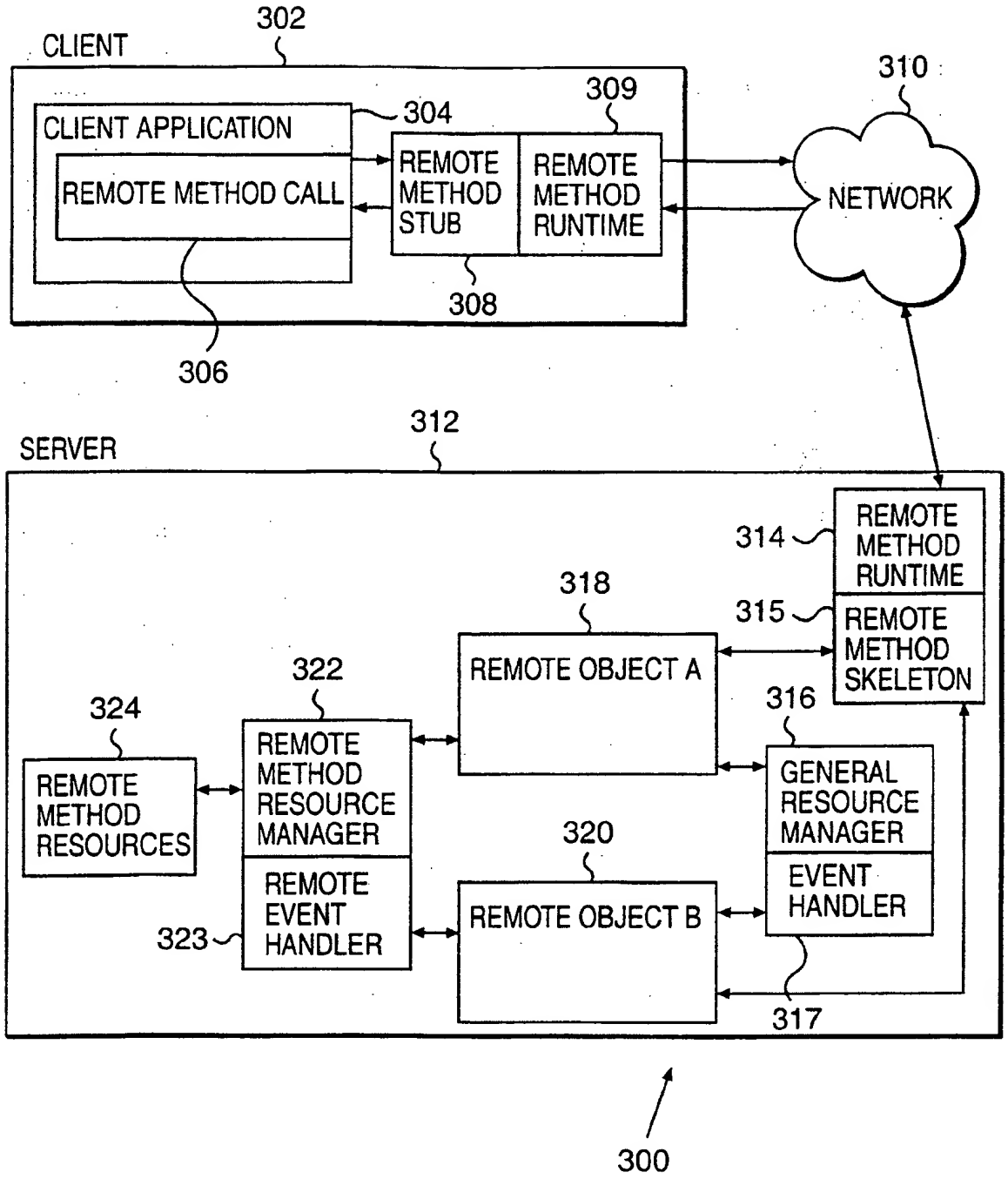


FIG. 3

BNSDOCID: <WO___9944131A1_I_>

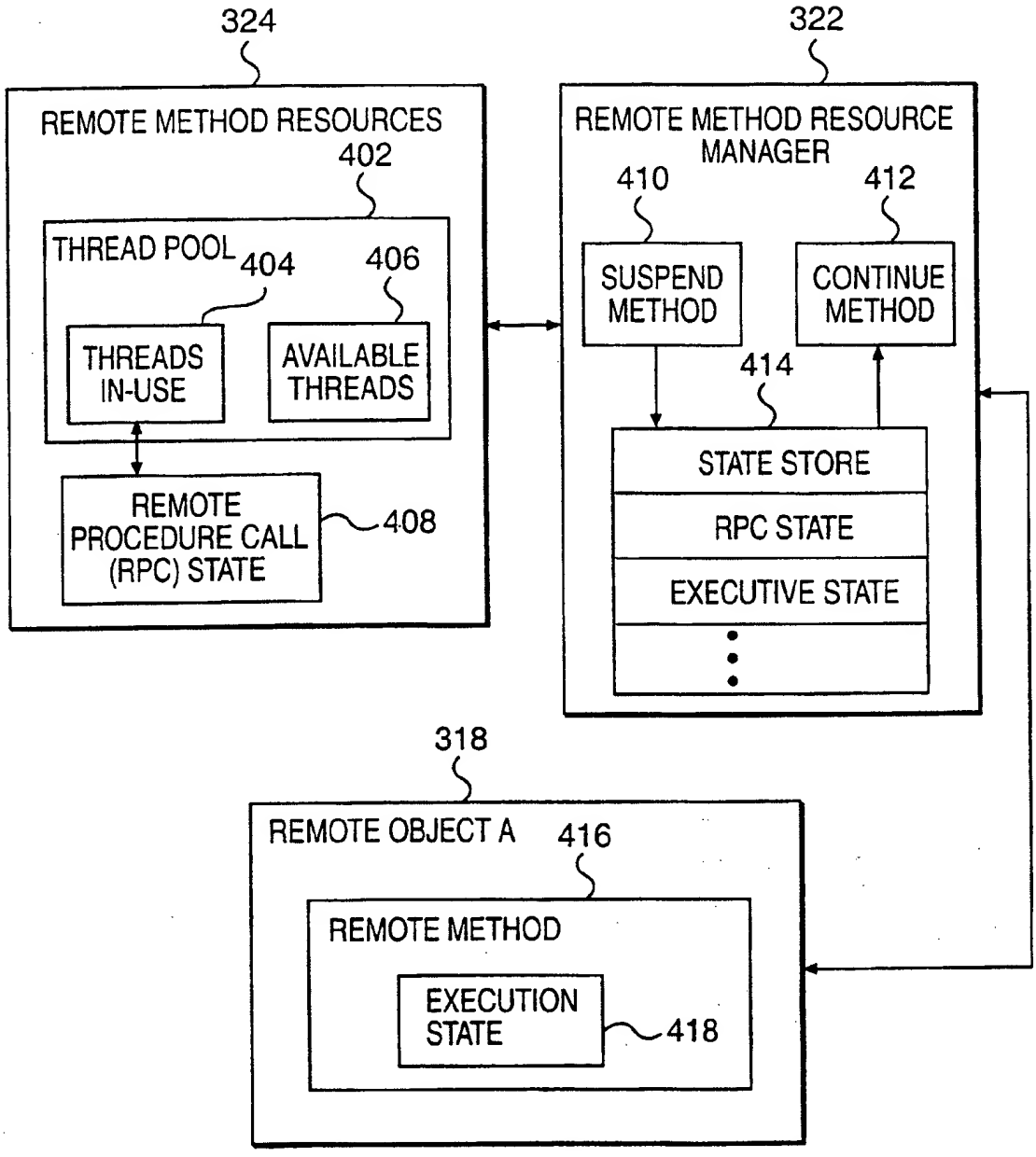


FIG. 4

BNSDOCID: <WO___9944131A1_I_>

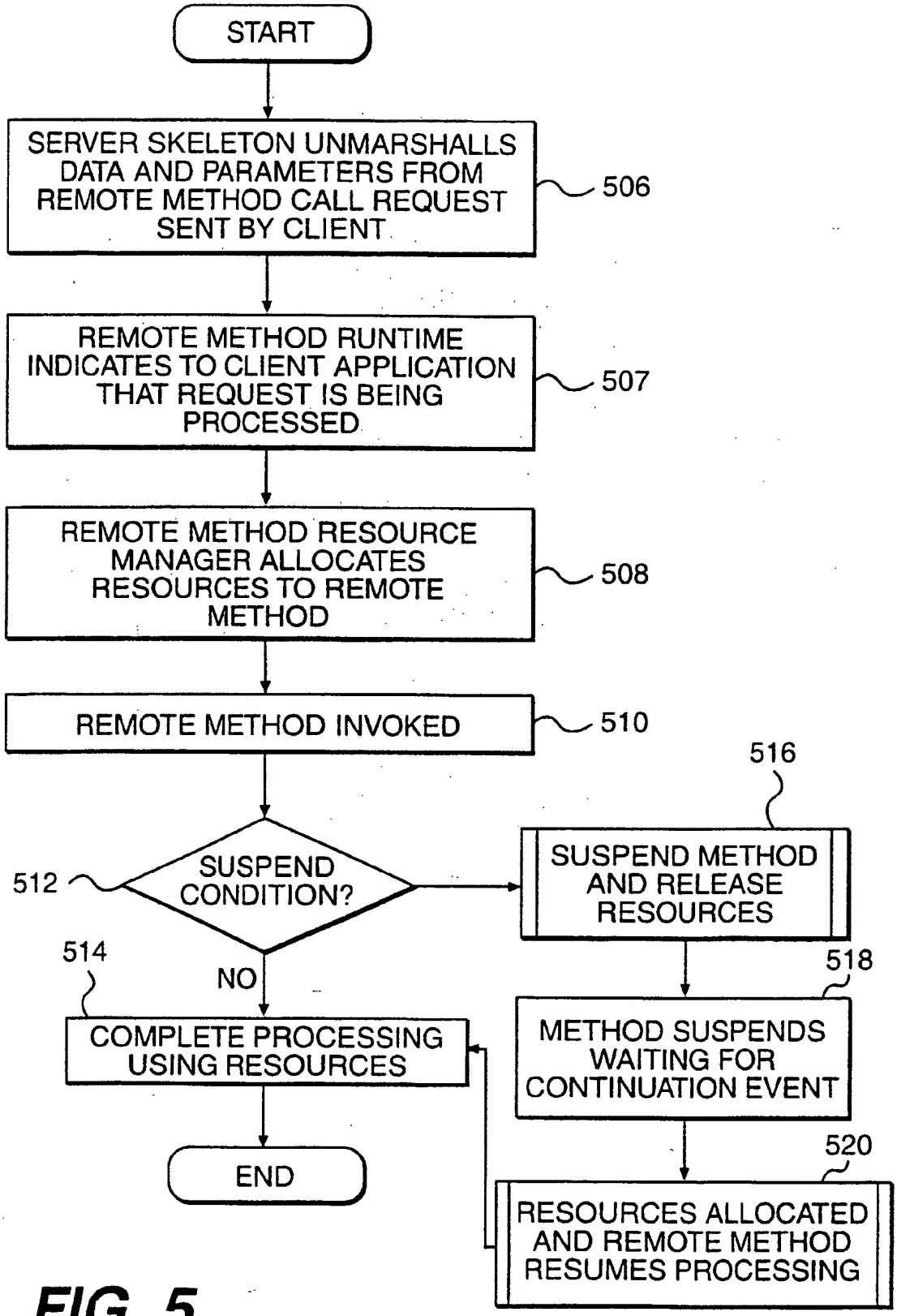


FIG. 5

BNSDOCID: <WO___9944131A1_I_>

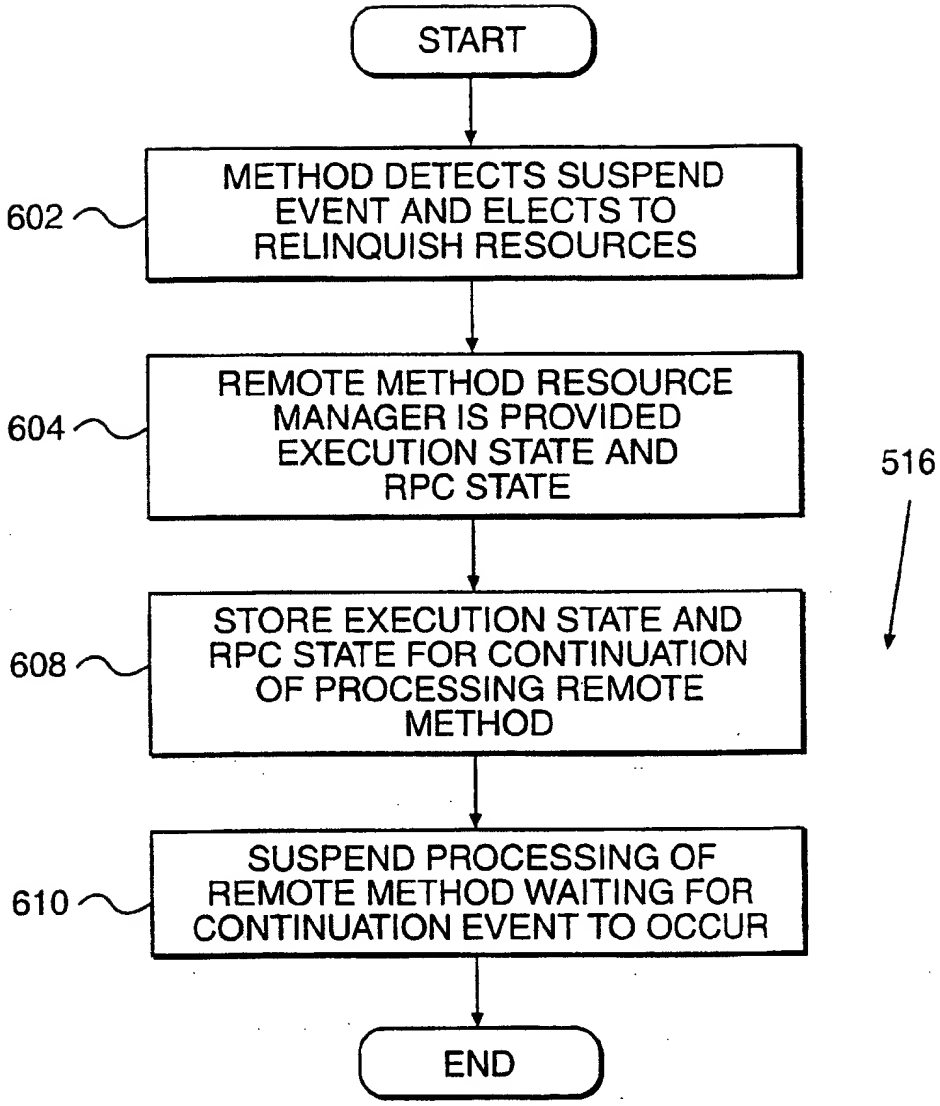


FIG. 6

BNSDOCID: <WO___9944131A1_I_>

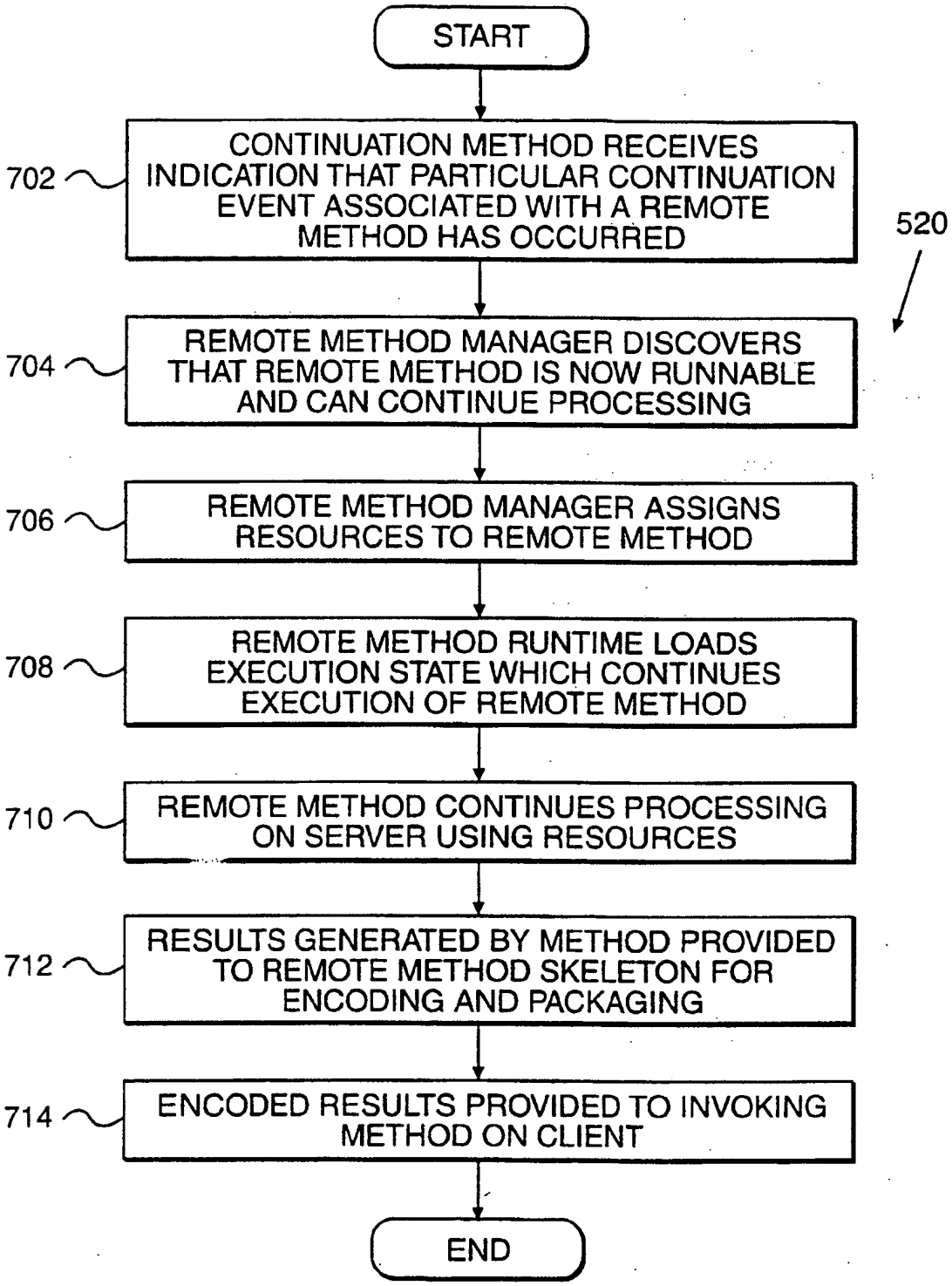


FIG. 7

SUBSTITUTE SHEET (RULE 26)

ENSDOCID: <WO 9944131A1 I >

INTERNATIONAL SEARCH REPORT

Int. ational Application No
PCT/US 99/03520

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 565 849 A (IBM) 20 October 1993 (1993-10-20) column 1, line 55 - column 2, line 25 column 23, line 11 - column 24, line 51 ---	1-39
A	WO 96 03692 A (BRITISH TELECOMM ;BUTT JOHN (GB); IRELAND PAUL STUART (GB)) 8 February 1996 (1996-02-08) page 5, line 28 - page 8, line 2 page 9, line 19 - line 32 page 11, line 15 - line 32 --- -/--	1-39

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
27 July 1999	10/08/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bijn, K

1

Form PCT/ISA/210 (second sheet) (July 1992)

BNSDOCID <WC__9944131A1_I_>

INTERNATIONAL SEARCH REPORT

Int. ..tional Application No
PCT/US 99/03520

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where apropriate, of the relevant passages	Relevant to claim No.
A	WO 92 07335 A (CRAY RESEARCH INC) 30 April 1992 (1992-04-30) page 11, line 19 - page 12, last last page 18, line 19 - page 20, line 26	1-6, 8-14, 16-18, 20-23, 25-28, 30-34
A	GB 2 305 087 A (FUJITSU LTD) 26 March 1997 (1997-03-26) page 20, line 12 - page 22, line 10 page 27, line 9 - page 28, line 7	1-4, 9-11,14, 16,21, 25,26, 31,32, 36-39

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

BNSDOCID: <WO..... 9944131A1 I >

INTERNATIONAL SEARCH REPORT

Information on patent family members

Intc. ..ional Application No
PCT/US 99/03520

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0565849 A	20-10-1993	US 5553305 A	03-09-1996
		JP 6012386 A	21-01-1996
WO 9603692 A	08-02-1996	AU 689925 B	09-04-1998
		AU 2986695 A	22-02-1996
		CA 2194766 A	08-02-1996
		EP 0772825 A	14-05-1997
		FI 970288 A	24-01-1997
		JP 10503306 T	24-03-1998
		NO 970315 A	24-01-1997
		NZ 289894 A	24-11-1997
		US 5889944 A	30-03-1999
WO 9207335 A	30-04-1992	AT 116456 T	15-01-1995
		DE 69106384 D	09-02-1995
		DE 69106384 T	10-08-1995
		EP 0553158 A	04-08-1993
		JP 6502033 T	03-03-1994
GB 2305087 A	26-03-1997	JP 9062526 A	07-03-1997
		US 5802298 A	01-09-1998

BNSDOCID: <WO___9944131A1_I_>

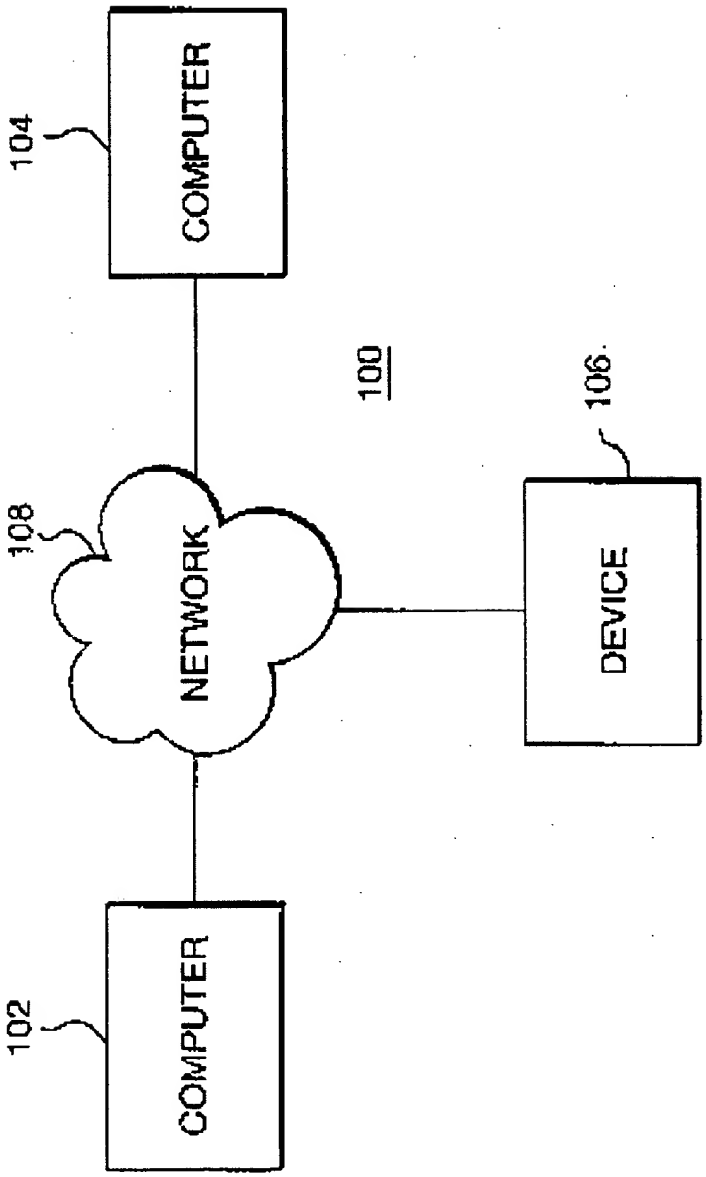


FIG. 1

BNSDOCID: <WO 9944131A1TI >

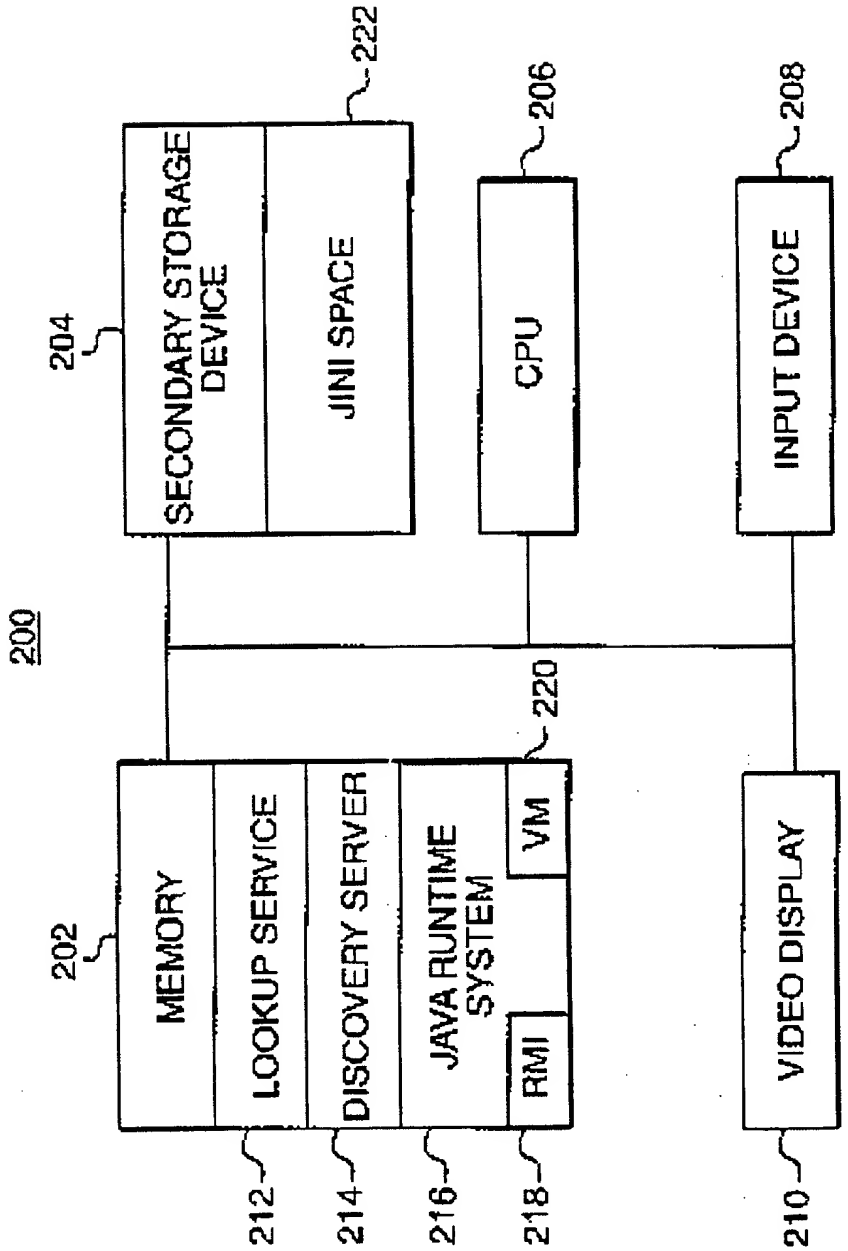


FIG. 2

BNSDOCID: <WO___9944131A1TI_>

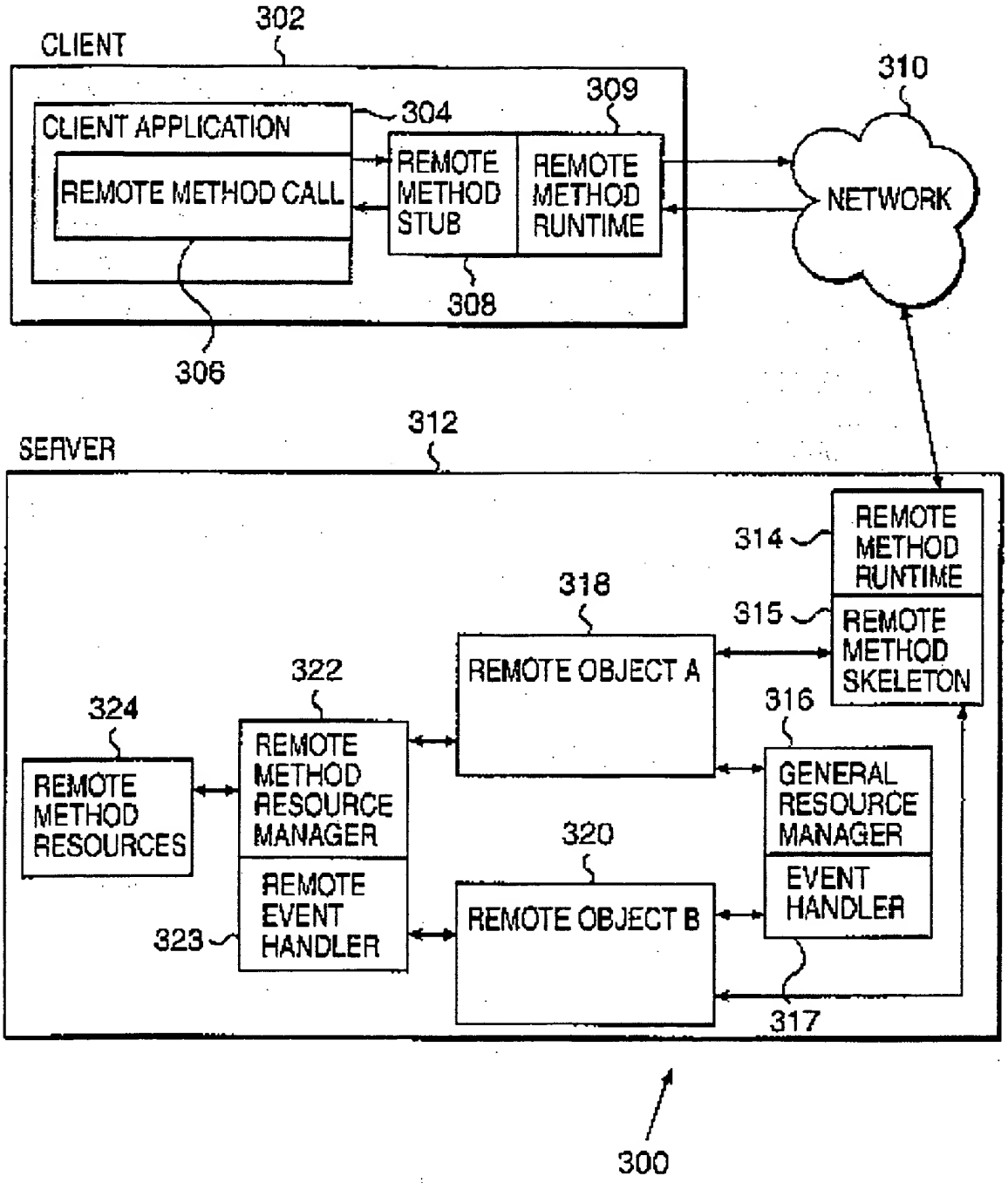


FIG. 3

BNSDOCID: <WO___9944131A1T1_>

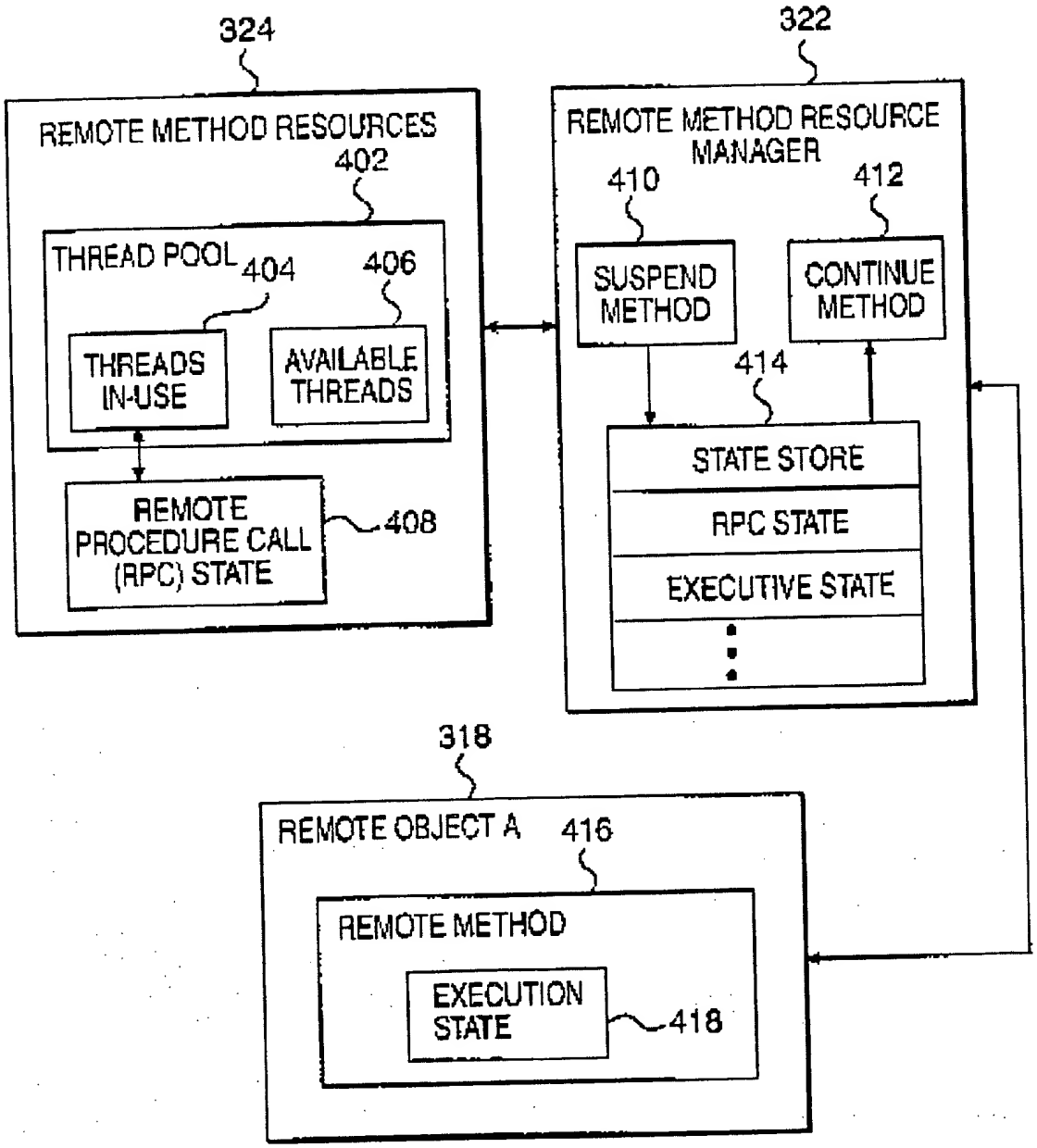


FIG. 4

BNSDOCID <WO___9944131A1TI_>

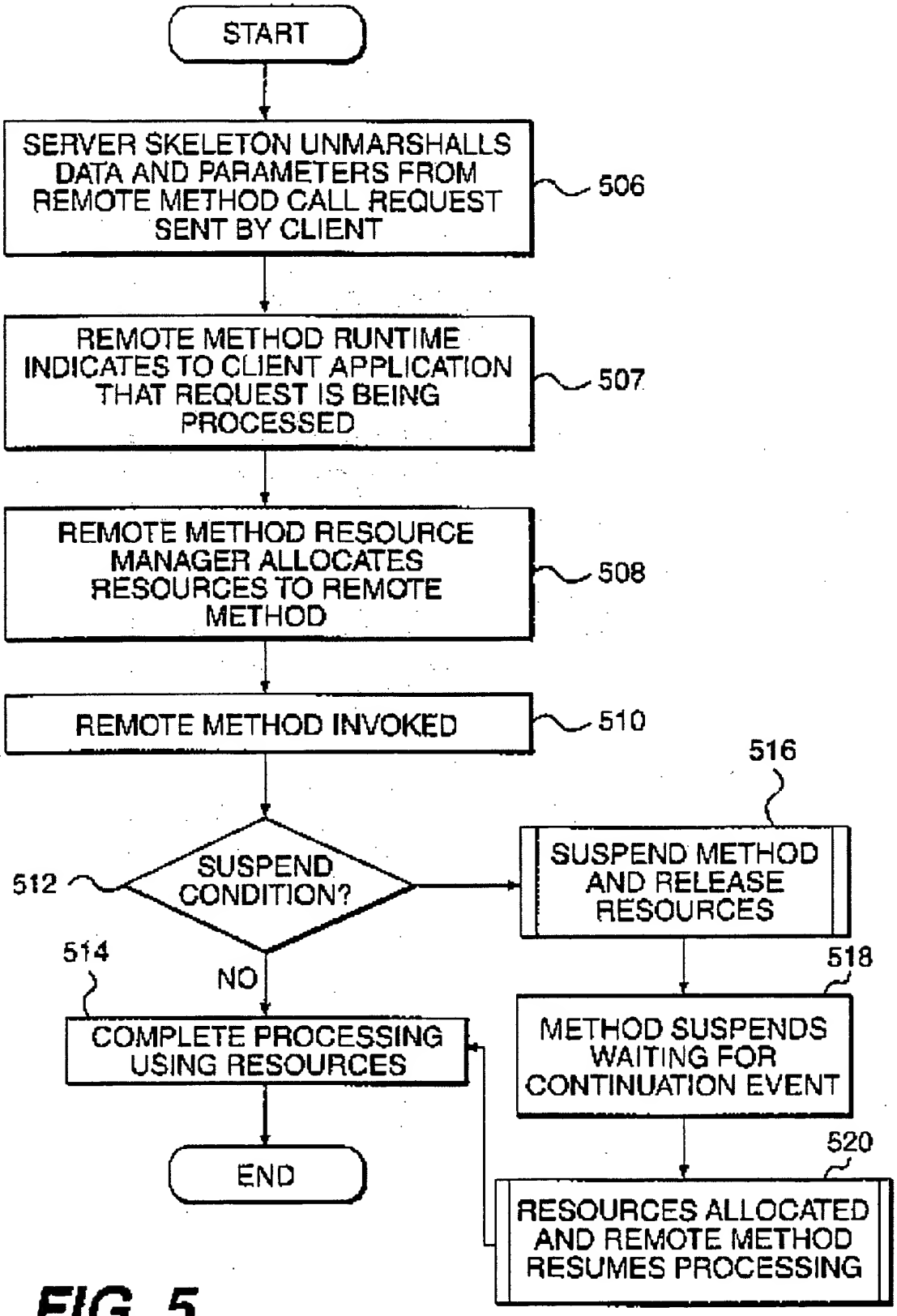


FIG. 5

BNSDOCID: <WO___9944131A1TI_>

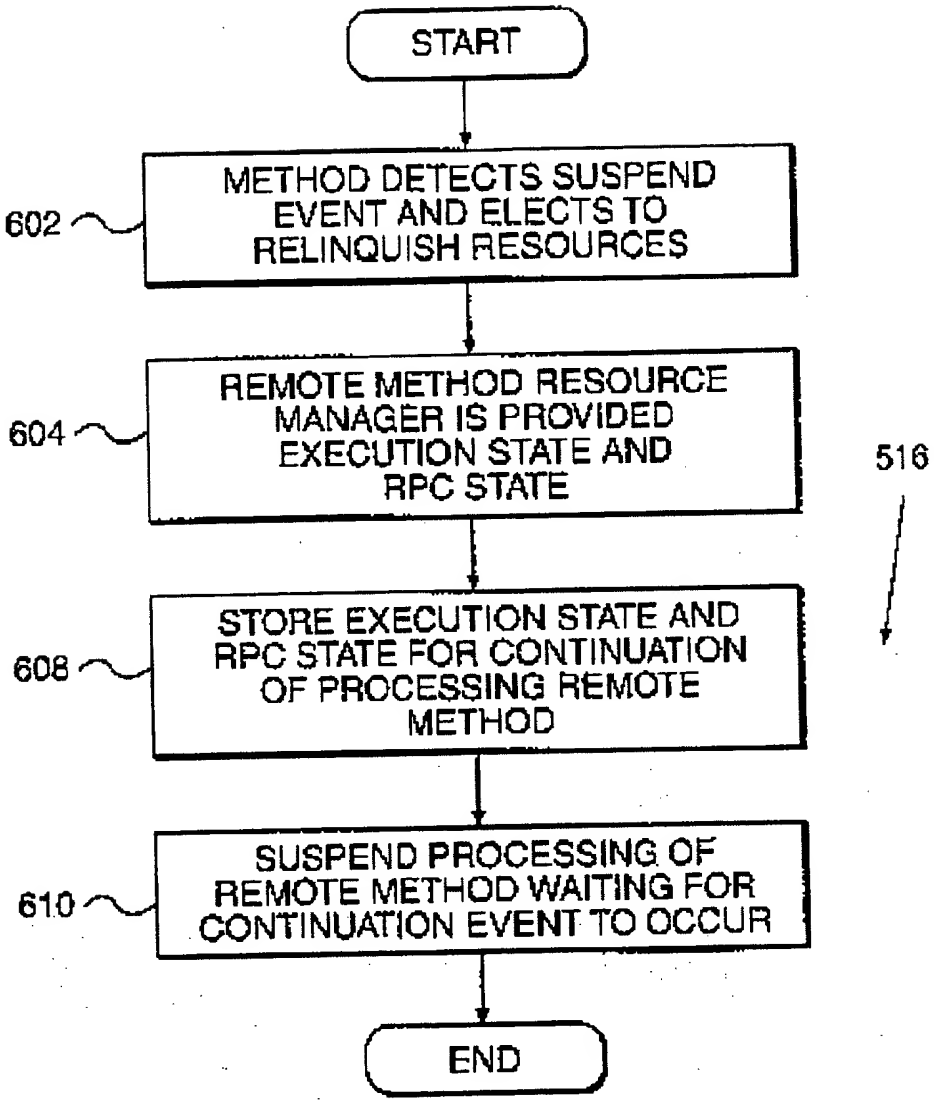


FIG. 6

BNSDOCID: <WO___9944131A1TI_>

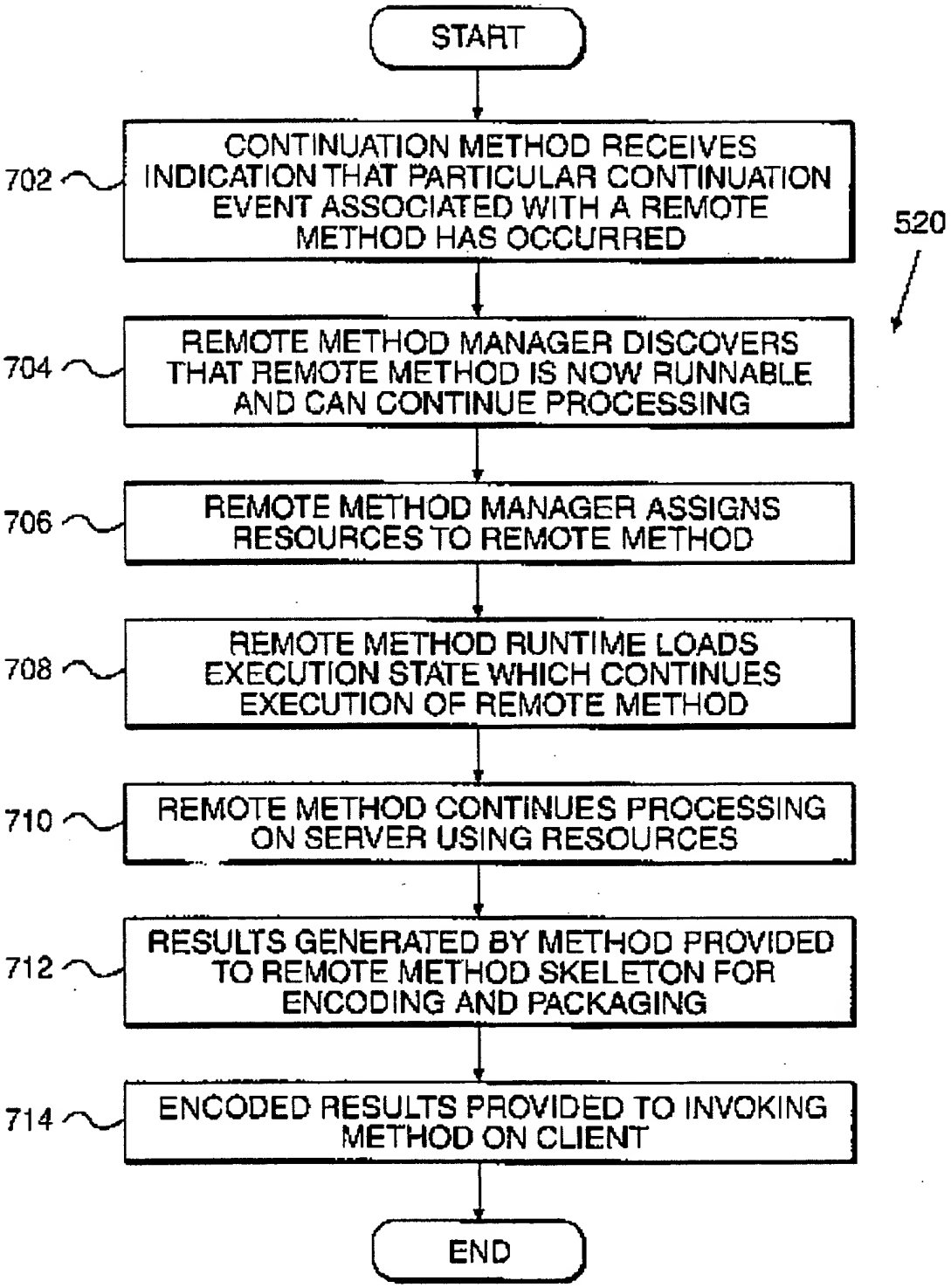


FIG. 7

BNSDOCID: <WO 9944131A1TI >

This Page Blank (uspto)